Entwurf - Transportband „Fabrik der Zukunft“

Scherlin, Erik

Braun, Florian

Großmann, Moritz

16. Mai 2017

Inhaltsverzeichnis

1 Systemumfeld

Dem Transportmanager ist eine Steuerung übergeordnet, welche für das Verwalten der einzelnen Subsysteme zuständig ist. Eines davon ist das hier beschriebene Transportsystem selbst, welches für den reibungslosen Transport zwischen den einzelnen Subsystemen verantwortlich ist. Die Kommunikation beider wird über eine Ethernetverbindung abgewickelt. Dabei sendet die Steuerung Befehle an den Transportmanager. Nach Erhalt der Nachricht wird der Befehl quittiert indem eine Bestätigung zurück an die Steuerung gesendet wird. Der Manager kümmert sich nun um das ausführen des Befehls und wird bei Abschluss des Auftrags eine abschließende Bestätigungsnachricht an die Steuerung zurücksenden. Das Bindeglied

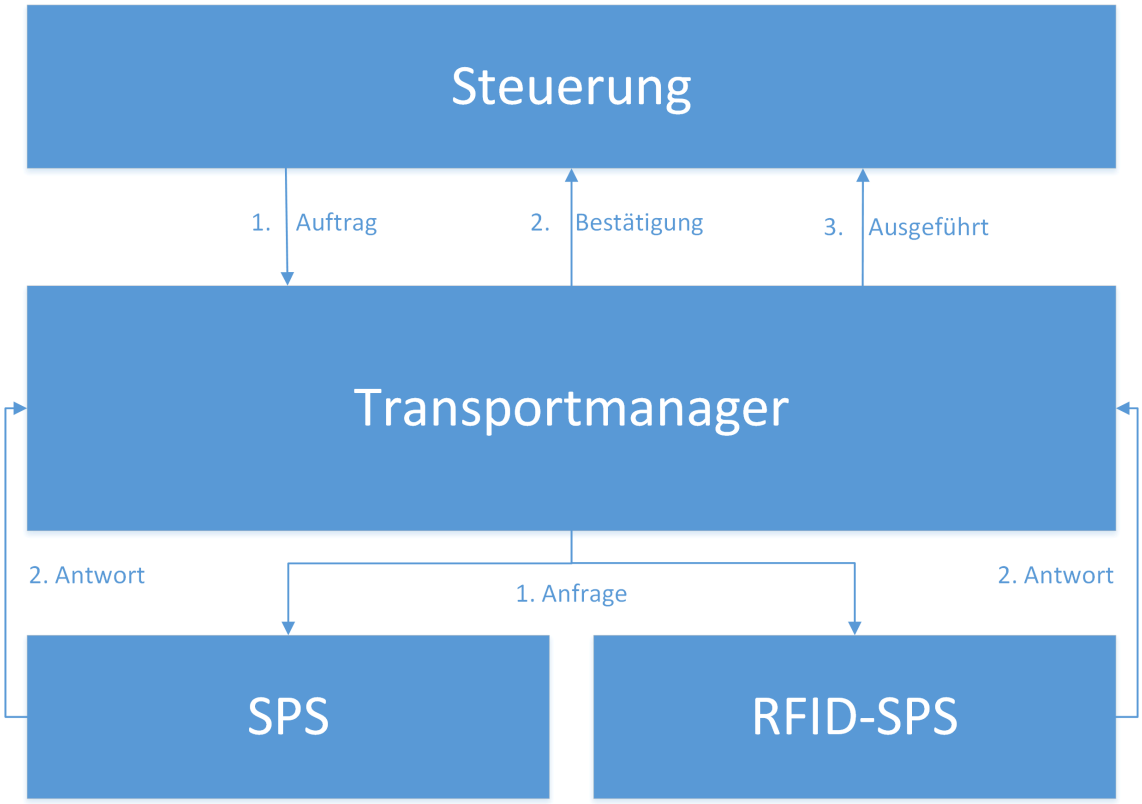

Abbildung 1: Übersicht über die Kommunikationswege zwischen den Subsystemen

zwischen dem Transportmanager und der Anlage bilden Speicherprogrammierbare Steuerungen (SPS). Die Kommunikationen finden hierbei ebenfalls über Ethernet statt. Die Haupt-SPS ist für das lesen und setzen von Sensoren sowie Aktoren verantwortlich. Übergestelle Komponenten stellen, mittels Request, eine Anfrage an das System und erhalten in Form eines gebündelten Pakets eine Auflistung der einzelnen Sensorwerte. Wohingegen die Aktoren nicht ausgelesen werden können. Diese werden mithilfe eines Befehls gesetzt und darauffolgend wird von der SPS eine Er-

folgsnachricht zurück gesendet. Für das Verarbeiten der RFID-Lesegeräte wurde in das System eine weitere SPS integriert, welche parallel zur Haupt-SPS läuft und Operationen durchführt. Hierbei können wieder mittels Requests, Anfragen an das System gestellt werden. Dadurch wird versucht, mithilfe des Angefragten RFID-Lesers, einen RFID-Chip auszulesen. Die gelesene ID oder der erfolglose Versuch zu lesen wird anschließend von der SPS an den Transportmanager zurückgemeldet.

2 Konzepte

Die besondere Herausforderung bei der Umsetzung der Software ist die Anforderung, dass mehrere Schlitten gleichzeitig auf dem Band transportiert und koordiniert werden sollen. Viele Arbeiten müssen dabei parallel passieren z. B. Kommandos entgegen nehmen und Schlitten routen. Zudem sind viele Dinge zeitlich nicht vorhersehbar, d. h. die Software weiß nicht wann ein Schlitten an einen Sensor kommt um diesen dann zu lesen. Aus diesem Grund wird die Software im Kern ereignisorientiert aufgebaut sein. Die Ereignisse werden von geänderten Sensordaten ausgelöst, die von einem eigenständigen Modul verarbeitet werden. Ein zweiter Trigger für die Planung der Routen und die Verwaltung der Schlitten sind die Kommandos, die die übergeordnete Steuerung sendet.

Der ereignisorientierte Ansatz bietet einerseits der Effizienz als auch beim Softwaredesign Vorteile. Die Sensordaten müssen zwar aufgrund von Einschränkungen bei der SPS zyklisch gelesen werden, allerdings wird nur die Verarbeitungslogik für die Sensoren angestoßen, die sich geändert haben. Da höherwertige Anlagenelemente (Weichen, Nodes, ...) intern Sensoren verwenden, werden diese auch bei einer Änderung des Zustands benachrichtigt und können ihre Logik ausführen. Da dies teils viel Zeit in Anspruch nehmen kann sollte in Erwägung gezogen werden, die Arbeitspakete auf Threads in einem Threadpool aufzuteilen. Die kritischen Methoden sollten deshalb provisorisch mithilfe eines Mutex geschützt werden.

Damit das Programm trotz der Verstrickung der Objekte übersichtlich und wartbar bleibt, sollen Abhängigkeiten durch Interfaces aufgelöst werden. Mit diesen werden die Schnittstellen beschrieben, die die konkreten Objekte bereitstellen. So kann als Referenztyp das Interface angegeben werden, das Objekt kann die Implementierung frei wählen. So ist auch der Austausch einzelner Komponenten einfach möglich, ohne dass tiefe Eingriffe in den Quelltext gemacht werden müssen.

Um auch den Aufbau der Anlage von der Logik, wie Dinge gemacht werden, zu trennen wird dieser in einer externen Beschreibungsdatei hinterlegt. Diese Datei ist als XML-Datei aufgebaut und besitzt Elemente für Bahnen, Sensoren, Stopper,

Weichen und Nodes. Zusätzlich zu den Tags können Informationen zu den Elementen angegeben werden, typischerweise eine ID mit der das Element später referenziert werden kann und weitere elementspezifische Informationen. Für jedes Element, dass in XML definiert werden kann, existiert ein Interface mit Setter-Methoden, die die möglichen Attribute entgegennehmen. Damit ist sichergestellt, dass das konkrete Objekt später diese Methoden anbietet und vom Parser befüllt werden kann. Es muss dazu lediglich das Interface implementieren. Außerdem ist so auch wieder die eigentliche Speicherung der Daten von der Beschreibung und dem Parsing getrennt.

3 Anlagenelemente

3.1 Physische Ebene

In dieser Ebene sind alle Teile der Anlage eingegliedert, die auch physisch vorhanden sind, also Aktoren und Sensoren.

3.1.1 Transportbahnen

Die Bahnen dienen der groben Beschreibung der Anlage. Sie besitzen lediglich eine eindeutige Identifikation um sie später in der Beschreibungsdatei und im Transportmanager zuordnen zu können. Sie sind außerdem ein elementarer Bestandteil der Routingstrategie. Sie werden in der Beschreibungsdatei aufgelistet aber im Programm nicht als Objekte gehandhabt, da Bahnen keine besondere Funktionalität bieten. Stattdessen sollen lediglich Strings mit der ID verwendet werden.

3.1.2 Metallsensoren

Diese Sensoren werden für die Erkennung von Schlitten verwendet. Jeder Sensor besitzt eine eindeutige Identifikation, mit der der Sensor in der Beschreibungsdatei referenziert werden kann. Sinnvollerweise wird hier der Name verwendet, der auch auf den Sensoren aufgeklebt ist. Die Übermittlung der Sensordaten von der SPS zum Programm geschieht nicht einzeln für jeden Sensor sondern als Byte-Array in dem ein einzelnes Bit jeweils für einen bestimmten Sensor gilt. Deshalb wird als zusätzliches Attribut die Position des relevanten Bits im Byte-Array benötigt.

3.1.3 Stopper

Mithilfe von Stoppern werden Schlitten angehalten, um sie an einer Station zu fixieren oder um nicht auf andere Schlitten aufzufahren. Als eindeutige Identifikation

besitzen diese eine frei wählbare Zeichenkette um den Stopper im Dokument zu referenzieren. Sinnvoll ist auch hier wieder der aufgedruckte Name auf der Anlage. Zusätzlich wird die Identifikation für die Kommunikation mit der SPS benötigt, mithilfe der Stopper ein- oder ausgefahren werden kann. Dieser Vorgang soll mithilfe von Sensoren automatisiert werden. Dafür müssen Informationen hinterlegt sein, welcher Sensor die Einfahrt beziehungsweise Ausfahrt eines Schlittens signalisiert, dafür werden die bereits vergebenen Identifikationen verwendet.

3.1.4 Weichen

Weichen spielen beim Routing der Schlitten eine entscheidende Rolle da sie verschiedene Bahnen und damit auch Stationen miteinander vernetzen. Zur Kommunikation mit der SPS wird eine Identifikation hinterlegt, mit der die Stellung verändert werden kann. Zusätzlich werden für das Routing Informationen benötigt, welche Transportbahnen mithilfe der Weiche erreicht werden können. Sie besitzt deshalb eine Auflistung mit Identifikationen der Bahnen und den Werten die zur SPS gesendet werden müssen, um die Weiche in die passende Stellung dafür zu bringen.

3.1.5 RFID-Leser

Die RDIF-Leser spielen bei der Erkennung von Paletten eine wichtige Rolle. Jede Palette besitzt eine eindeutige Nummer, die auf einem RDIF-Tag abgespeichert ist. Diese Nummer wird für den Transport zur richtigen Station benötigt. Die Leser dienen aber auch der Erkennung leerer Schlitten, sobald ein Metallsensor ein Signal gibt und das Lesen einer Palettennummer nicht möglich ist muss es sich um einen leeren Schlitten handeln. Um mit der SPS kommunizieren zu können muss eine Identifikation hinterlegt werden.

3.2 Logische Ebene

3.2.1 Knotenpunkte

Um ein Routing der Schlitten durchzuführen zu können müssen an bestimmten Punkten auf der Strecke Entscheidungen getroffen werden, in welche Richtung der Schlitten fahren muss um sein Ziel zu erreichen. Diese werden als Knotenpunkte, oder Englisch „Nodes“, bezeichnet. Zusätzlich werden auch die Endstationen, also Lager, Roboter, E/A-Station und der Leerschlittenpuffer als Nodes geführt. Dies vereinheitlicht die Beschreibung der Anlage und die Software. An den Nodes wird ein einfahrender Schlitten überprüft und „identifiziert“ um festzustellen, was sein

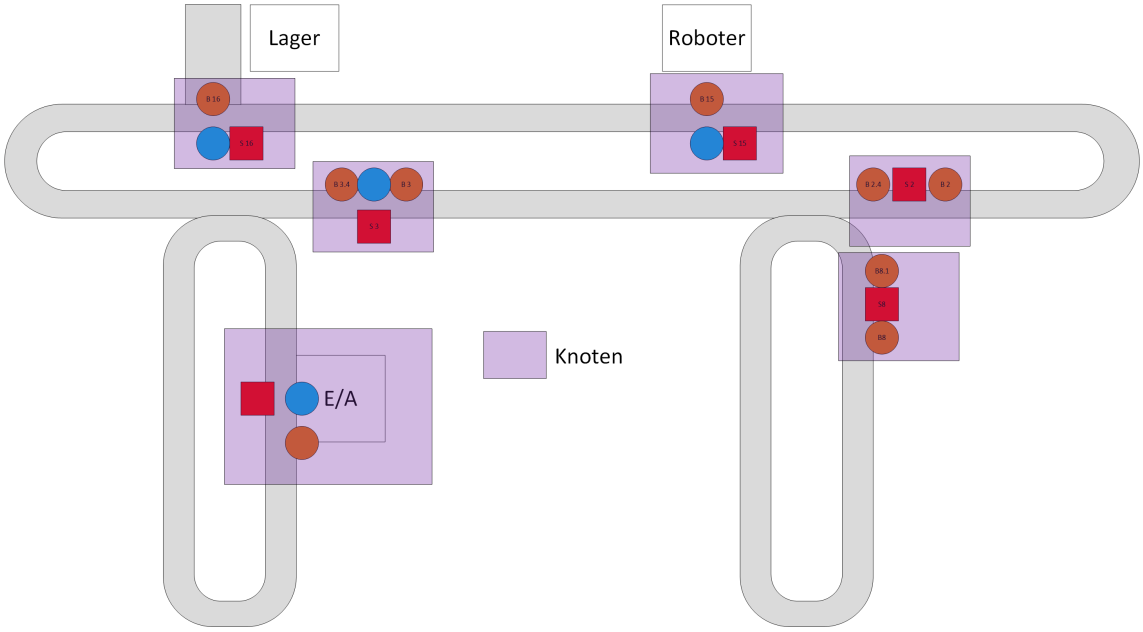

Abbildung 2: Übersicht über die Knotenpunkte der Anlage

Ziel ist. Wenn auf dem Schlitten eine Palette liegt kann über deren RFID-Tag die zugehörige Destination ermittelt und der Schlitten dorthin gebracht werden. Ist an einem Knoten kein RFID-Leser verfügbar wird die aktuelle Palette über die Informationsweitergabe zwischen den Knoten ermittelt. Handelt es sich um einen leeren Schlitten wird als Ziel der Knoten verwendet, der als erstes einen Schlitten angefordert aber noch nicht bekommen hat.

Knotenpunkte besitzen zur Informationsweitergabe eine Queue, in der die demnächst vorbeifahrenden Schlitten hinterlegt werden. Bei einem Routingvorgang wird mithilfe des Zielknotens die passende Route ermittelt wodurch auch der nächste Knoten, der angefahren wird, bekannt ist. Der Schlitten wird dabei aus der eigenen Queue entfernt und in die des nächsten Knotens eingefügt.

4 Modulübersicht

Um die Abhängigkeiten im Projekt und im Programm so gering wie möglich zu halten wird der Transportmanager in unabhängige Module aufgeteilt. Die Schnittstellen, über die auf die einzelnen Module zugegriffen werden kann, werden vorher mit Interfaces festgelegt. Damit soll auch eine einfachere Entwicklung mit mehreren Personen möglich werden. Abbildung 3 zeigt die einzelnen Module.

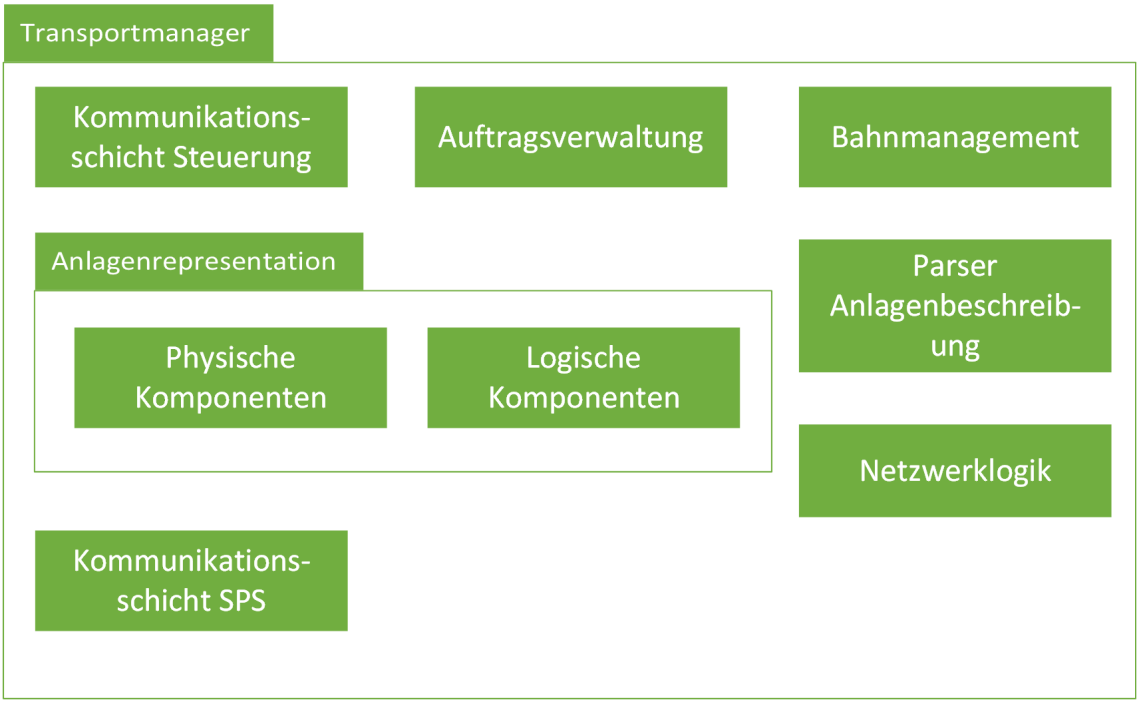

Abbildung 3: Übersicht über die Module im Transportmanager

5 Modulbeschreibung

5.1 Kommunikationsschicht zur Steuerung

Um Transportaufträge bearbeiten zu können müssen diese vom übergeordneten System, der Steuerung, entgegengenommen und verwaltet werden. Diese Aufgabe soll von der Kommunikationsschicht zur Steuerung erledigt werden. Dafür muss diese beim Starten eine Verbindung zur Steuerung aufnehmen und aufrecht erhalten über welche dann die Anfragen und Antworten versendet werden. Dies soll in einem eigenen Thread geschehen, der nur für die Netzwerkkommunikation verantwortlich ist. Damit wird sichergestellt, dass durch lange Laufzeiten beim Verwalten der Transportaufträge keine Netzwerkpakete verloren gehen.

Da Transportaufträge nicht sofort bearbeitet und quittiert werden können müssen diese zwischengespeichert und verwaltet werden. Dazu sollen Informationen wie die eindeutige ID, die Auftragsart und das Ziel gespeichert werden. Die ID wird verwendet um abgeschlossene Aufträge an die Steuerung zurückzumelden und aus dem Management zu entfernen. Die anderen Informationen werden für das Routing benötigt. Sobald von der Steuerung ein neuer Transportauftrag ankommt, wird dieser in die Liste der offenen Aufträge gespeichert und alle interessierten Objekte benachrichtigt. Dazu gehört das Bahnmanagement, das ebenfalls Teil der Schicht ist. Dieses

kümmert sich um die Schlitten und Paletten auf der Bahn und deren Position. In ihr sind auch alle Queues zu den einzelnen Nodes gespeichert. Das Bahnmanagement entscheidet auch welche Schlitten für welchen Auftrag verwendet werden. Es stellt außerdem Schnittstellen nach außen bereit, damit Nodes Informationen zu ihren Queues und Routen abfragen können.

5.2 Kommunikationsschicht zur SPS

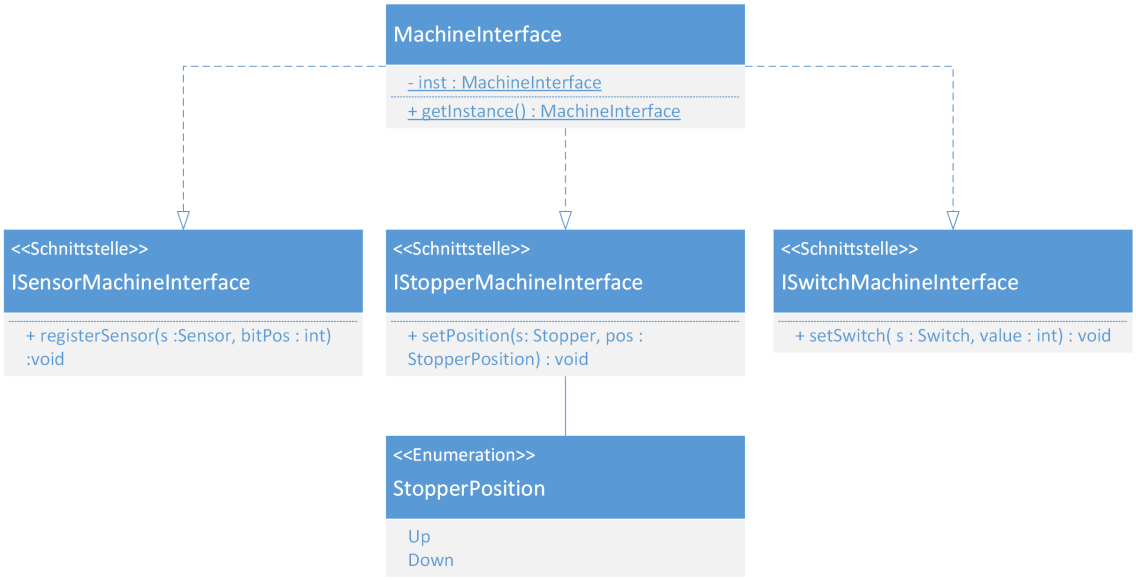

Abbildung 4: Schnittstellen zum Zugriff auf die Hardware

Um die Funktionalität von der eigentlichen Implementierung zu trennen wird die Anlage, besser gesagt die SPS, mithilfe von Interfaces beschrieben. Dabei wird für jeden Typ von Hardware eine Schnittstelle definiert.

Da die SPS von sich aus keine Pakete sendet, wenn sie der Wert eines Sensors geändert hat wird in einem eigenen Thread zyklisch der Zustand aller Sensoren abgefragt. Das Ergebnis ist dabei ein Byte-Array bei dem ein einzelnes Bit für einen Sensor steht. Das Interface *ISensorMachineInterface* (siehe Abbildung 4) definiert eine Methode, mit der man einen Sensor beim Benachrichtigungsmechanismus registrieren kann. Benötigt werden das Sensorobjekt, um dieses bei einer Änderung zu benachrichtigen, und die Position des relevanten Bits im Array. Die Angabe erfolgt dabei in der natürlichen Schreibweise eines Arrays. Angenommen man hat ein Array *a* mit der Länge 3, dann befindet sich das Bit 0 in *a[0]*, das Bit 8 in *a[1]* usw.

Für die Interaktion mit Stoppern wird das Interface *IStopperMachineInterface* (siehe Abbildung 4) eingeführt. Es definiert eine Methode mithilfe der Zustand eines

Stoppers verändert, also der Stopper ein- oder ausgefahren, werden kann. Identifiziert wird der Stopper mithilfe seiner ID bei der SPS, die in der Beschreibungsdatei hinterlegt ist. Für die Positionsangabe steht eine Enumeration zur Verfügung.

Die Funktionalität einer Weiche wird vom Interface *ISwitchMachineInterface* (siehe Abbildung 4) definiert. Diese kann in verschiedene Stellungen gefahren werden, je nachdem welches Band bedient werden soll. Um die Schnittstelle möglichst abstrakt und allgemein zu halten wird hier nur die ID der Weiche in der SPS und der Wert zum Setzen der Position übergeben. Diese werden in der Beschreibungsdatei hinterlegt. Somit sind in Zukunft auch andere Weichenarten einfach zu implementieren.

Die drei Schnittstellen werden mit der Klasse *MachineInterface* (siehe Abbildung 4) zur Verfügung gestellt. Intern besitzt sie eine Socketverbindung zur SPS, die permanent aufrecht erhalten wird. Zudem besitzt sie für das Abgreifen der Sensordaten einen Thread, der zyklisch die Daten abholt. Das Benachrichtigen der Sensoren geschieht von einem anderen Thread aus, da ansonsten Sensordaten verloren gehen könnten wenn die Verarbeitungszeit sehr lang ist. Dieser Thread ist damit auch der Trigger für alle Aktionen die im Programm stattfinden, da ausgehend vom Sensor auch die anderen Komponenten und Strategien gestartet werden.

5.3 Kommunikationsschicht zur RFID-SPS

Die Kommunikationsschicht zu dieser SPS knüpft direkt an dem in ... beschriebenen Konzept an. Liest die SPS die Tags nur auf Anfrage oder wird automatisch gelesen und dann gespeichert?

5.4 Parser für Anlagendefinition

Der Transportmanager soll in sich modular aufgebaut sein, d. h. die einzelnen Module sollen möglichst unabhängig voneinander sein. Der Transportmanager soll aber auch von der zu steuernden Anlage getrennt werden. Dafür soll die Logik, *wie* transportiert wird, vom genauen Aufbau der Anlage unterschieden werden. Diese Trennung erleichtert die Anpassung des Programms an neue Anlagenkonfigurationen oder Änderungen am Aufbau, außerdem können so einfach neue Sensoren und Wegpunkte eingeführt werden ohne Codeänderungen vorzunehmen. Der Testaufwand wird so deutlich reduziert.

Abstrakt gesehen lässt sich die Anlage mithilfe von wenigen Elementen beschreiben. Auf der untersten Ebene sind die Sensoren und Aktoren der Anlage eingegliedert. Eine Abstraktionsebene höher befinden sich logische Knoten die für das Routing der Schlitten verwendet werden. Die Elemente der einzelnen Ebenen sind in

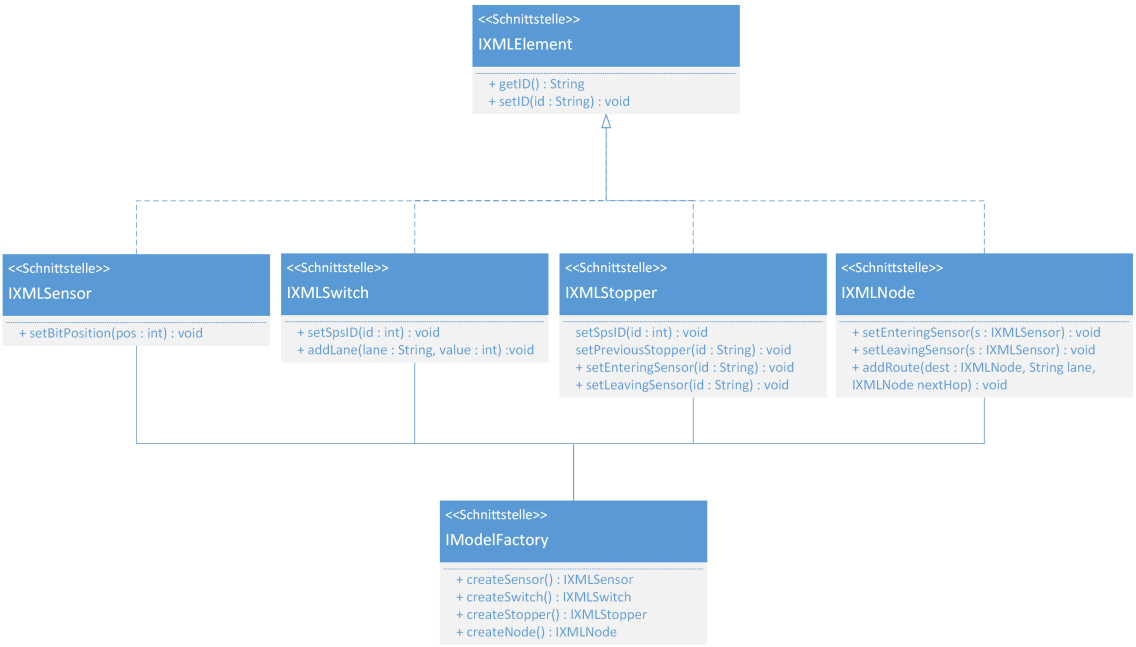

Abbildung 5: Übersicht über die Schnittstellen in der XML-Komponente

Kapitel 3 erläutert. Zu jedem Element existiert ein Interface, das Setter-Methoden definiert, mit denen die Daten der XML-Datei verarbeitet werden können. Diese dienen der Entkoppelung der Komponenten. Es werden jedoch konkrete Objekte benötigt die je nach Implementierung unterschiedlich sein können. Der Parser definiert dazu ebenfalls ein Interface *IModelFactory* in dem Methoden vorgeschrieben werden, die eine Instanz eines bestimmten Typs zurückliefern. Die Instanz dieser Fabrik wird dem Parser im Konstruktor übergeben. Dadurch ist die Verantwortung, welche Klassen verwendet werden, in das Hauptprogramm übergegangen. Zusätzlich können in der Fabrik anlagenspezifische Informationen gesetzt werden, die nichts mit dem Parsevorgang zu tun haben.

Um alle Abhängigkeiten möglichst einfach aufzulösen geht der Parser immer nach einem festen Schema vor, um die Elemente zu laden. Die Transportbahnen werden dabei nicht gesondert betrachtet, können aber zur Validierung herangezogen werden.

Als erstes werden die Sensoren (siehe Kapitel 3.1.2) geladen, da diese keine anderen Komponenten benötigen.

5.5 Auftragsverwaltung

Die Auftragsverwaltung beinhaltet für jede gestellte Anfrage einen eigenen Auftrag, welcher beim Empfangen einer Anfrage angelegt wird. Diese beinhaltet eine eindeutige ID, Ein Auftrag wird innerhalb der Verwaltung in eine Warteschlange

eingereiht. Diese wird von oben nach unten abgearbeitet. Dies gewährleistet den zeitlichen Ablauf der einzelnen Anfragen und dient zugleich als Priorisierung der Aufträge. Wurde der Auftrag erledigt, wird dieser abgeschlossen. Dazu gehört das Entfernen des Auftrags aus der Warteschlange, sowie das Rückmelden des Erfolgreichen Abschlusses an die Steuerung. Letzteres erfolgt mittels der in ... beschriebenen Kommunikationsschicht Steuerung.

5.6 Transportlogik

5.7 Thread-Management

Da im Transportmanager mehrere Sachen gleichzeitig ablaufen müssen, werden Threads benötigt. Um einen möglichst geringen bei der Verwaltung der Threads zu haben wird diese Funktionalität in einen Threadpool ausgelagert.

In der Klasse *Poolthread* steckt die Logik zum Abarbeiten von Jobs. Sie ist als Nested-Class in der Threadpool realisiert, sie kann deshalb auch auf private-Methoden des Threadpools zugreifen. Die Jobs werden im Threadpool verwaltet, die Liste mit Jobs kann mithilfe der Methode *getJobs* abgerufen werden. Diese ist nicht als synchronized definiert da hier nur die Referenz auf eine Liste zurückgegeben wird, die sich nicht ändert.

Neue Jobs können über die Methode *addJob* an den Pool übergeben werden. Ist momentan ein Thread frei fängt dieser mit der Abarbeitung ab, ansonsten wird der Job später abgearbeitet sobald ein anderer fertiggestellt wird. Dabei wird ein Thread benachrichtigt der dann wieder aufgeweckt wird.

Der Poolthread selbst prüft in seiner Abarbeitungslogik endlos ob ein Job bearbeitet werden muss. Wenn ja holt er sich diesen aus der Liste und löscht ihn gleichzeitig, dieser Vorgang geschieht atomar. Wenn nein geht er mit *wait* in den wartenden Zustand.

5.8 Netzwerkkommunikation

Der Transportmanager ist das netzwerktechnische Bindeglied zwischen der Steuerung und beiden SPS-Systemen.

Die Bestandteile der Netzwerksschicht setzen sich folgendermaßen zusammen:

SPS-Kommunikationsmodul Schnittstelle zur SPS; ließt Sensordaten zyklisch aus und sendet Aufträge zum setzen von Aktoren.

Steuerung-Kommunikationsmodu l Empfängt Aufträge von der Steuerung und reit diese in den Abarbeitungspuffer ein

Parser Anlagenbeschreibung Ist für die Anlagendefinition verantwortlich ist und erzeugt zum Programmstart alle Module sowie Abhängigkeiten untereinander.

Das Modul der SPS-Kommunikation besteht aus zwei Teilen:

SPSCommunication Bei der Instanziierung stellt diese Klasse eine Socketverbindung zur SPS her und hält diese aufrecht. Zur Kommunikation stehen Methoden zum senden und empfangen von Paketen zur Verfügung.

SPSPackageHandler Ein dauerhaft laufender Thread, welcher für das Verwalten der SPS-Kommunikationsschicht verantwortlich. Die Klasse wird als Singelton gehandhabt, um die Einmaligkeit zu gewährleisten und das bereitstellen der Methoden an die Logische Ebene zu gewährleisten. Instanziierung: Bei der Instanziierung durch den Parser wird auch die Instanz der Klasse SPSCommunication erzeugt, welche der PackageHandler inne hält. Ebenfalls speichert er das Mapping der einzelnen Sensoren als Liste. Die Einträge bestehen aus den einzelnen Sensorobjekten, welches benachrichtigt werden sollen und der jeweils zugehörigen Position im Byte-Array, welche beim Anfragen der Sensordaten zurückgegeben wird. Zyklischer Ablauf: Anschließend liest der Thread zyklisch die Sensorwerte aus. Dafür stellt er eine Anfrage an die SPS und erhält alle Werte gebündelt als Byte-Array. Das erhaltene Array wird zwischengespeichert und eine Überprüfung findet statt. Dazu wird das aktuell gelesene Array und das des letzten Zyklus herangezogen und auf Abweichungen überprüft. Sollte eine Abweichung auftreten wird eine Benachrichtigungsroutine gestartet. Benachrichtigungsroutine: Sobald eine Abweichung auftritt wird ein Thread aus dem Thread-Pool geholt, welcher für die weitere Abarbeitung zuständig ist. Dies hat den Vorteil, dass die Benachrichtigungsroutine den zyklischen Lesethread nicht blockiert und so keine Wertänderungen verpasst werden. Der neue Thread sucht sich den gemappten Sensor aus der Liste heraus und benachrichtigt diesen. Hierfür steht das Interface *ISensorStateChanged* zur Verfügung, welches für die Benachrichtigung verwendet wird. Hierdurch wird eine Benachrichtigungskette angestoßen. Der Sensor, welcher Benachrichtigt wurde, wird nun alle darauf registrierten Objekte antriggern (Bsp. Weichen, Stopper, o.ä.), welche dann Ihre Routinen wieder Separat ausführen werden (Bsp. Stellen der Weiche, benachrichtigen weiterer Objekte, o.ä.). Nach dem Abarbeiten der Prozedur wird der Thread wieder verworfen, d.h. er geht zurück an den Thread-Pool und kann wieder neu vergeben werden. Setzen von Aktoren:

Für das Setzen von Werten steht eine Schnittstelle zur Verfügung, welche über die Instanz (Singleton) einfach bedient werden kann. Über die Methode addJob() wird ein Job der Liste hinzugefüt. Dieser enthält die in der SPS registriete ID und den zu setzenden Zustand. Nun wird nach jedem Zyklusdurchlauf alle angefallenen Job's abgeabertet. Hirzu wird einfach mittels sendPackage() eine Anfrage mit den jeweiligen Attributen an die SPS versendet. Abschließend wird der Job aus der Liste wieder entfernt. (Ohne Job Liste und der Thread versendet die Nachricht direkt?)

Das Modul der Steuerungs-Kommunikation besteht aus zwei Teilen:

ControlCommunication Bei der Instanziierung stellt diese Klasse eine Socketverbindung zur SPS her und hält diese aufrecht. Zur Kommunikation stehen Methoden zum senden und empfangen von Paketen zur Verfügung.

ControlPackage Ähnlich, wie bei der Klasse *SPSPackageHandler* handelt es sich hierbei um einen Thread und das Singelton Pattern wird verwendet. Instanziierung: Die Instanziierung findet hier ebenfalls durch den Parser statt und die Instanz der Klasse ControlCommunication wird auch erzeigt. Ebenfalls stellt dieser den Bezug zur Auftragsverwaltung her. Nach der Instanziierung wird auf den erhalt einer Nachricht gewartet. Anfrage von Auftrag: Sobald ein Auftrag eingeht wird dieser quittiert, d. h. es wird eine Bestätigung an die Steurung zurückgesendet, dass die Nachricht erfolgreich erhalten wurde. Anschließend wird - wie oben - ein Thread aus dem Thread-Pool gezogen und in die Auftrags Abschließen eines Auftrags: Wurde ein Auftrag von der Transportlogik erfolgreich abgeschlossen wird dies nochmals an die Sterung weitergeleitet. Hierzu holt sich die logische Komponente die Singelton-Instanz der Klasse und ruft die Methode finishJob(jobID) auf. Die Methode sendet nun eine Bestätigungsnachricht an die Steuerung, dass der Job mit der ID jobID erfolgreich abgearbeitet wurde.